Analysis of Paradoxes in Fingerprint Countermeasures

Andalibi, Vafa

FRUCT Oy

2017-11-10

Andalibi , V , Christophe , F & Mikkonen , T 2017 , Analysis of Paradoxes in Fingerprint Countermeasures . in Proceedings of the 21st Conference of Open Innovations Association FRUCT, University of Helsinki, Helsinki, Finland . FRUCT Oy , pp. 398-401 , Conference of Open Innovations Association , Helsinki , Finland , 06/11/2017 .

http://hdl.handle.net/10138/232259

Analysis of Paradoxes in Fingerprint Countermeasures

Vafa Andalibi
Indiana University Bloomington
Bloomington, USA
vafandal@indiana.edu

Francois Christophe, Tommi Mikkonen
University of Helsinki
Helsinki, Finland
{francois.christophe, tommi.mikkonen}@helsinki.fi

***Abstract*—The widespread usage of new user tracking methods, i.e. web-based fingerprinting, is becoming a serious privacy concern as third parties try to track users across different websites. Meanwhile, it is usually difficult or impossible for users to opt-out fingerprinting if they want to fully benefit the services provided by the application or website. Several studies tried to address the privacy issue in browser fingerprinting, mostly by faking attribute values. However, such configuration spoofing may lead to inconsistencies that paradoxically make the user stand out even more. This study analyzes these paradoxes in browser configuration with the creation of a Markov model based on a test dataset. Given a target spoofed attribute, the implemented tool in this study outputs the other attributes that must be consequently altered, not to cause paradoxical configuration. Similarly, this tool can suggest a set of random attributes to be spoofed with suggested values, not creating a paradoxical configuration. The tool Implemented in this study can be used by browser extension developers and should help them spoof browser attributes more sophistically, thus preserving users' privacy against cross-site web-based browser fingerprinting.**

I. Introduction

Cookie is the asset for storing information corresponding to session state, e.g. user's personal configurations as well as private data, on client device. The main purpose for cookies is to identify users and track their past activities on a website for their convenience and rapidity of access. Due to the functional nature of cookies, it is always a privacy and security concern in modern web browsers and so it is possible to disable cookies. Nevertheless, a new method of tracking users was recently proposed by anti-fraud and advertising companies which employs fingerprinting.

As the large-scale experiments conducted in [1], [2] shows, web-based fingerprinting is a serious privacy concern. There are two main reasons behind the motivation for device fingerprinting:

Third-party tracking, i.e. tracking users across unrelated websites to build up a user profile allowing efficient and accurate advertisement targeting. Fraud prevention, i.e. web-based fingerprinting is a powerful tool for finding related transaction. The gathered database can be used to blacklist fraudulent users. Moreover, illegal sharing of accounts can be detected using fingerprinting.

The fingerprint of a device is a combination of system attributes that can be queried and accessed from the browser. This combination has usually a high likelihood to be unique and therefore can be used as a device identifier. A natural best practice for web-based fingerprinting is to select attributes based on their stability, i.e. attributes that seldom change or at least change very gradually. These attributes, generally collected via JavaScript, range over a broad set of values such as, the User agent header, the Accept header, the Connection header, the Encoding header, the Language header, the list of plugins, the platform, the cookies preferences (allowed or not), the Do Not Track preferences (yes, no or not communicated), the time zone, the screen resolution and its color depth, the use of local storage, the use of session storage, a picture rendered with the HTML Canvas element, a picture rendered with WebGL, the presence of AdBlock, and the list of fonts [1].

Although opting-out is possible from third-party fingerprinting services, there is indeed no guarantee for a successful opt-out. Even for giving the possibility of a successful out-out, the fingerprint needs to be computed anyway, assuming cookies are disabled and that the third-party is honest. A common approach in building browser extensions for fingerprint opt-out is to induce a shared fingerprint, so that all the users with that fingerprint are not distinguishable [3].

During the data gathering phase of Panopticlick project[1], Eckersley found that impossible configurations are reported by the browsers with spoofed configuration. For instance, such impossible configuration would be a device claiming to be an iPhone and supporting flash plug-ins at the same time. Using common browser features was shown to be effective in Eckersley's study, therefore providing extensions to increase privacy added by users is reasonable. As a case study, members of a cybercriminal forum were advised to change user-agent by installing and utilizing an extension for anonymity [4].

Nikiforakis et al. [5] analyzed the most popular browser extensions that are purposed to spoof *user-agent*. In their test, *navigator* and *screen* objects, as two most-probed objects by the fingerprinting libraries, were listed via JavaScript and subsequently compared with HTTP header sent with their request. Interestingly, they found that in *all* cases, the true identity of the browser, inadequately hided by

ISSN 2305-7254

the extension, is exposable through JavaScript straightforwardly and many of them were suffering from the impossible configuration issue. Particularly, they mention that *none* of the analyzed extensions try to modify the *screen* object. Therefore, impossible screen resolution was reported by all the users who were browsing on a laptop or workstation and disguised it as a smartphone, e.g. 4k resolution on an iPhone.

In this work, we address impossible configurations, i.e. paradoxes in the configuration resulted from alteration of browser extensions. A dependency model of the browser features is created based on the attributes that are used for device fingerprinting. This Markov chain model implicitly contains the dependencies between the attributes by simple terms of probabilities. Hence, this Markov model reveals two facts about the afore-mentioned paradoxical example. First, it highlights that alterations of user-agent must be followed by changing the display resolution. Second, a possible screen resolution that matches the chosen user-agent, e.g. 750x1334, is recommended to the developer. Consequently, web-based device Fingerprinting countermeasures can follow this model to report consistent spoofed features.

The structure of this paper is as follows. First, the related studies and configuration spoofing tools and extensions are presented. This is followed by the methodology and subsequent results of the study. The evaluation of the current tool is finally presented in the last section.

II. Related works

Since the main purpose of this study is to help to build a successful countermeasure to device-fingerprinting, the related works presented in this section are focused on countermeasures. Moreover, web-based device fingerprinting was presented in introduction.

Many browser plugins are developed with the main goal of stopping trackers, e.g. Ghostery, AVG Do Not Track, User-Agent Switcher for Chrome, etc. Main academic fingerprinting countermeasure studies includes FireGloves [6], ShareMeNot [7], PriVaricator [8]. The defense of some of these works are based on blacklisting the trackers. This approach is not much effective for two reasons: first, not all trackers are covered in black listing, second, some trackers can try to obfuscate their scripts, bypassing the blacklisting filter. Similarly for heuristically updated blacklist, any new tracker that implements a method that is not covered by that particular heuristic may not be noticed, as presented in [9], Mowery et al. describe how to use the blacklist content itself as an additional fingerprinting element.

Other works, e.g. FireGloves, User-Agent Switcher for Chrome, and PriVaricator, try to fake the attributes which as pointed out in [1], [5], usually lead to inconsistencies and paradoxes that make the user stand out even more. This corresponds to the issue addressed in this study. This case is specifically examined in [10] for browsers employing FireGloves. Nikiforakis et al. [5] suggest not to use any user-agent-spoofing browser extension but this is in our opinion rather extreme and the following section presents our method allowing the use of spoofing without inducing paradoxes.

III. Design and implementation

The dataset analyzed in this study is a MySQL database provided by AmIUnique fingerprinting project website (www.AmIUnique.org). This dataset has 16000 records containing over 30 fingerprinting attributes for the devices fingerprinted during March 2017, including id, addressHttp, time, acceptHttp, hostHttp, userAgentHttp, pluginsJS, connectionHttp, encodingHttp, languageHttp, orderHttp, platformJS, cookiesJS, fontsFlash, IEDataJS, timezoneJS, resolutionJS, localJS, sessionJS, resolutionFlash, rendererWebGLJS, webGLJs, languageFlash, platformFlash, adBlock, octaneScore, sunspiderTime, pluginsJSHashed, dntJS, canvasJSHashed, webGLJsHashed, and fontsFlashHashed. The analysis of this dataset is implemented in python.

To model the dependency in the fingerprinting attributes, a Markov model of the data was first created in the learning phase, which is saved as a python dictionary in a pickle object for further use. During the learning phase, first, each of the *n* columns of the dataset was analyzed separately and possible unique values were find.

In the next step, with respect to rows containing each of the unique values, all the remaining n-1 columns where analyzed to find the values that are possibly linked with them. However, two main issues remain with this approach. First, this task can be extremely resource intensive (NP hard) and optimization of some sort is required. Second, the model follows a n-factorial nested loop as, for instance, for every variable in any of the n-1 columns that were previously bound to the unique variables of the first column, the same process should be repeated for values of the other n-2 columns and so on.

The former issue is caused by the fact that there might be many rows in data that are unique. These unique rows are not useful for our model for two reasons. First, since these rows are unique, the Markov chain will eventually contain nodes that are very unlikely to happen and this is counter intuitive for this work since the goal is to generate a reasonable and at the same time generic configuration of attributes. Second, while not useful, these rows significantly decrease the performance of the model.

To address the first issue, a *frequency threshold* value was defined during modeling. This value defines the minimum frequency of each of the values for a particular column. In implementation of this study, this frequency threshold was set to 30. In other words, there must be at least 30 instances of a particular value for an attribute to be considered and consequently saved as part of this model. Frequency threshold will inevitably cause skipping several records with lower frequencies. Therefore, the sum of final probabilities of possible values of an attribute might not sum up to 100%. The probabilities of values therefore had to be normalized to overcome this issue.

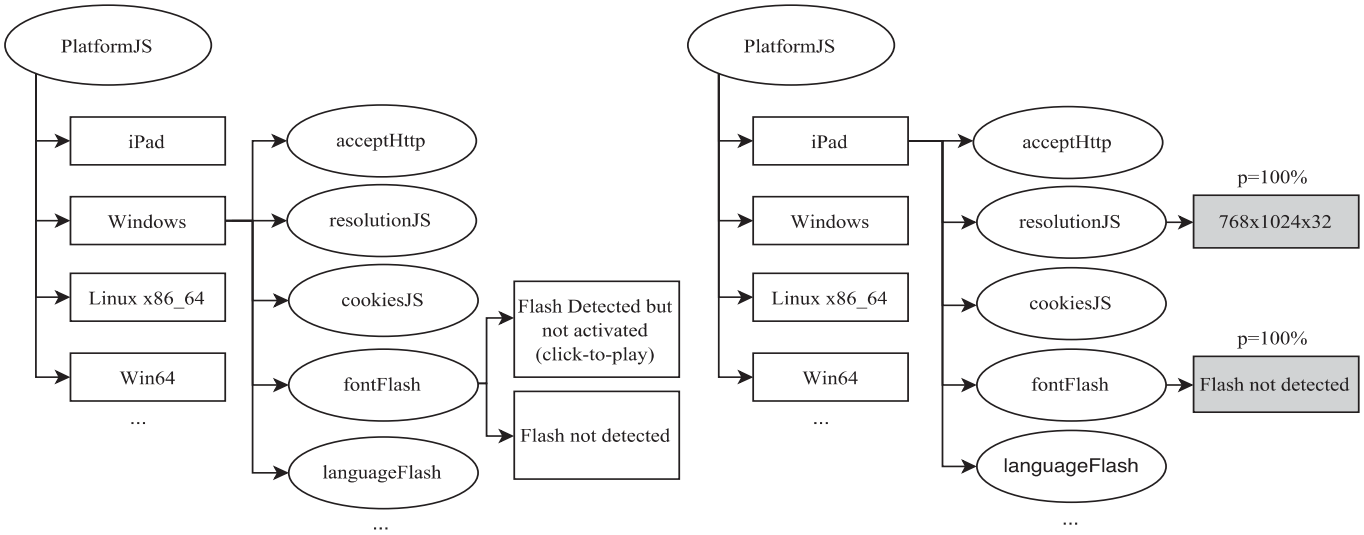

Fig. 1. example of spoofing the platformJS attribute to values would/would not cause paradox. Left: changing the value of platformJS to windows allows two choices for fontFlash attribute. Right: changing the value of platformJS to iPad, one must change the resolutionS and fontsFlash to 768x1024x32 and Flash Not Detected respectively, otherwise it would cause a paradox.

Regarding the second issue, i.e. repeating the process n! times, intuitively the model depth should be increased for the model to be more precise. However, the main objective of this work is not to generate a detailed and accurate model, but to generate a generic and reasonable one. Hence, the depth of the model was set to 2 and the first layer would again be used as the farther depths. This way, the method will still be realistic while increasing the performance.

Another major parameter that was applied on the method was dependency threshold. This parameter was applied and used after building the model during the data extraction phase. Suppose the goal is to determine whether for a value V1, from attribute A1, there is any variable V2 from attribute A2 that is dependent to it. This dependency, if exists and not addressed, will still cause the reporting of a near-unique attribute in system's configurations. To determine the dependency, the probabilities of all possible values of V2 will be considered and compared with dependency threshold. If the probability is higher than the *dependency threshold*, variables V1 and V2 will be considered as dependent.

Listing 1. output of the program, recommending a configuration for spoofing

```
Direct parameter: cookiesJS -> yes
Direct parameter: IEDataJS -> no
Direct parameter: connectionHttp -> close
Direct parameter: sunspiderTime ->
Direct parameter: sessionJS -> yes
Direct parameter: vendorWebGLJS -> Microsoft
Direct parameter: encodingHttp -> gzip, deflate
Direct parameter: octaneScore ->
Direct parameter: hostHttp -> amiunique-backend
Direct parameter: localJS -> yes
Direct parameter: platformJS -> Win32
Indirect parameter: vendorWebGLJS -> adBlock -> no
```

The implemented tool can also recommend a set of configuration changes that cause a reasonable spoofing. This task is achieved by walking on the Markov model. As a change in an attribute's value might lead to a paradox in another attribute, for each change, the consistency of other attributes is examined and indirect paradoxical attributes are determined and reported with suggested values for further spoofing.

IV. Results

Fig. 1 illustrates two spoofing: suppose a user intends to spoof the platformJS attribute to hide his browser's real fingerprint. By changing the platformJS to windows (Fig. 1 left), we notice that the fontFlash should also change to either of *Flash Detected But Not Activated (Click-to-play)*, and *Flash Not Detected*. Any value except these values is a paradox in browser's configuration. In the second example (Fig. 1 right), the platformJS is spoofed to iPad. Clearly, iPad's browser does not support flash and this was one of the paradoxes that was presented in [5] and here we see that with a probability of 100%, both the resolutionJS and fontFlash must be changed to 768x1024x32 and Flash Not Detected respectively.

In a similar manner, this tool can recommend a reasonable configuration for spoofing. Recommended configuration has some direct attributes and few indirect ones. Listing 1 illustrates an example of such configuration. In this listing, some of the parameters are left without recommended values. This is due to building up the model based on a small dataset of 16000 records for over 30 attributes. Hence, we assume that training the program with a larger dataset would fix this issue. This issue can also be fixed by reducing the frequency threshold followed by a performance decrease in learning phase.

VI. Evaluation and conclusion

In this study, an analysis of the paradoxes in web-browser fingerprinting countermeasure was presented. A

spoofing-helper tool based on a Markov chain of a sample dataset was also implemented in python. This script can work on the two following modes. First, given a value for a particular attribute, it will inform the developer about the other attributes that must be changed for consistency. Second, the script can suggest a set of changes that if applied, can successfully spoof the configuration of the browser.

The contribution of this work can be questioned by a high-security browser like TOR browser, that keeps the attribute of all its instances the same, hence making the fingerprinting ineffective. However, this only happens in ideal case where the users do not install any plugins and do not try to tweak the settings. Moreover, this strong privacy of TOR browser as a fingerprinting countermeasure comes at the expense of severe negative effect on usability and in many exceeds the use of non-oppressed users.

Although this script is mainly purposed to be used as a complementary tool for spoofing against fingerprinting, it can also be used for detecting paradoxes as well. For instance, one can input this tool with a configuration of a browser, and by extracting the probability of its attributes given the value of others, the total probability of the configuration can be calculated. If the configuration lacks consistency and is naïvely spoofed, the paradoxes will cause the total probability to be zero. Of course, for the result to be accurate a large dataset is required that contains many of possible combinations of the attributes.

As an interesting future work, the presented tool in this study can be extended and implemented as a browser extension which can re-identify the users who spoofed their configuration to avoid fingerprinting, i.e. linking a new fingerprint to an existing one due to lack of sufficient modification of the attributes.

ACKNOWLEDGMENT

We would like to thank the team of researchers at AmIUnique.org who generously provided us with one of their latest fingerprinting databases.

REFERENCES

[1] P. Eckersley, “How unique is your web browser?,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6205 LNCS, pp. 1–18.

[2] P. Laperdrix, W. Rudametkin, and B. Baudry, “Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints,” in *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, 2016, pp. 878–894.

[3] “Tor: Cross-Origin Fingerprinting Unlinkability.” [Online]. Available: https://www.torproject.org/projects/%0Atorbrowser/design/#fingerprinting-linkability.

[4] A. Klein, “How Fraudsters are Disguising PCs to Fool Device Fingerprinting,” *6June2012*, 2012.

[5] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, “Cookieless monster: Exploring the ecosystem of web-based device fingerprinting,” in *Proceedings - IEEE Symposium on Security and Privacy*, 2013, pp. 541–555.

[6] K. Boda, Á. M. Földes, G. G. Gulyás, and S. Imre, “User tracking on the web via cross-browser fingerprinting,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7161 LNCS, pp. 31–46.

[7] F. Roesner, T. Kohno, and D. Wetherall, “Detecting and defending against third-party tracking on the web,” *Proc. USENIX Conf. Networked Syst. Des. Implement.*, no. Nsdi, p. 12, 2012.

[8] N. Nikiforakis, W. Joosen, and B. Livshits, “Privaricator: Deceiving fingerprinters with little white lies,” in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 820–830.

[9] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham, “Fingerprinting Information in JavaScript Implementations,” *Web 2.0 Secur. Priv.*, pp. 1–11, 2011.

[10] G. Acar *et al.*, “FPDetective: Dusting theWeb for Fingerprinters,” *Proc. 2013 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS ’13*, pp. 1129–1140, 2013.